an online complaint form accessible from the course web page.

Solutions to homeworks and labs are posted online each week, in the directory `~cs61a/solutions` and on the course web page, the day after each assignment is due. You should definitely read these! They discuss most problems in some depth, with alternate solutions and suggestions for thinking about similar problems. Project and exam solutions are also posted, two days after the due date of each project, and whenever we finish grading each exam (because the posted solutions include a discussion of common wrong answers, which we don't know until we grade them).

For copies of handouts from lecture, look in the handout rack just inside the door of 283 Soda.

There is an electronic bulletin board system that you can use to communicate with other 61A students. To do this, subscribe to the `ucb.class.cs61a` newsgroup using any news reader, such as trn, Mozilla, or Thunderbird. You must specify `news` as the news server for the `ucb` newsgroups. On the machines in the lab, or other Unix systems, you can read the newsgroup by saying

```
rn -q ucb.class.cs61a
```

The `ucb` newsgroups can be read only from machines in the `berkeley.edu` domain, so if your net connection is through a commercial ISP then you must log into a lab machine to read the newsgroup or try this:

```
http://www-inst.eecs/connecting.html#news
```

If you post to the similarly-named newsgroup at Google Groups, your message will not be seen by those who read the actual on-campus group!

Please do not send electronic mail to every student individually! That would waste a lot of disk space, even for a small message. Use the newsgroup instead. Electronic mail is for messages to individuals, not to groups.

There is a class web page, with online versions of some of the documents we hand out:

`http://www-inst.eecs.berkeley.edu/~cs61a`

The web page for the textbook, with additional study resources, is

`http://www-mitpress.mit.edu/sicp/sicp.html`

There are also web pages for the Scheme programming language:

`http://www.swiss.ai.mit.edu/projects/scheme/index.html`
`http://www.schemers.org/`

Tutoring services are provided by Eta Kappa Nu (HKN), the EECS honors society (345 Soda, `hkn@hkn`), and Upsilon Pi Epsilon, the Computer Science honors society (346 Soda, `upe@cory`).

Additional information to help you in studying, including hints from the course staff and copies of programs demonstrated in lectures, is available on the course web page.

Homework and Programming Assignments

Each week there will be problems assigned for you to work on, most of which will involve writing and debugging computer programs. You'll work on some of these problems individually, and some in groups. These assignments come in three categories:

• **Laboratory exercises** are short, relatively simple exercises designed to introduce a new topic. Most weeks you'll do these during the scheduled lab meetings the first half of the week, in groups of two to four students.

• **Homework assignments** consist mostly of exercises from the textbook; you'll do these whenever you can schedule time, either in the lab or at home. You may be accustomed to textbooks with huge numbers of boring, repetitive exercises. You won't find that in our text! Each assigned exercise teaches an important point. These assignments are included in the course reader. (The first week's assignment is also attached to this handout.) Each week's assignment will be due Monday of the *following* week. You are encouraged to *discuss* the homework with other students, but each of you must prepare and turn in your own solutions. (More on this later.)

• **Projects** are larger assignments intended both to teach you the skill of developing a large program and to assess your understanding of the course material. There are four projects during the semester. The first two projects will be done individually; the last two in groups of two students.

Most of the weekly assignments include problems labelled as "Extra for Experts." These problems are entirely optional; do them only if you have finished the regular assignment and want to do something more challenging. There is no extra *credit* for these problems; people who need more credit shouldn't even be trying them, and people who are doing well in the course should be motivated by the desire to learn.

The purpose of the **homework** is for you to learn the course, not to prove that you already know it. Therefore, the weekly homeworks are not graded on the correctness of your solutions, but on effort. **You will get full credit for an entirely wrong answer that shows reasonable effort!** (But you should test your work. If your solution is incorrect, the grader will want to see some evidence that you know it's incorrect.)

Each homework is worth two points for a reasonable effort, zero points for a missing homework or one that seems to show no effort, or **negative four (-4) points** for a solution copied from someone else.

The four programming **projects** *are* graded on correctness, as well as on your understanding of your solution. The first two projects will be done individually; the last two in groups of two students, but some of the work is split up so that each problem is done by one student. You must work together to ensure that both group members understand the complete results. Projects must be turned in both online and on paper (see below).

Copying someone else's work does not count as "reasonable effort"! This includes copying from your friend who took the course last semester as well as copying from other current students. You will get negative credit for copied solutions, and repeated offenses will lead to more severe penalties. If you don't know how to do something, it's better to leave it out than to copy someone else's work.

You should try to complete the *reading* assignment for each week **before** the lecture. For example, you should read section 1.3 of the textbook this weekend, before the lecture of Monday, 1/25. (Read section 1.1 as soon as possible this week!) You will have five class meetings (three lectures and two discussion/lab sections) to help you understand the assignment. During that time you should begin to work on the exercises, and you should try to complete them in the lab during the second half of the week. If you're efficient, you'll then have the weekend to read the following week's reading assignment.

Each week's homework assignment will be turned in by noon Monday of the following week (or noon Tuesday if Monday is a holiday). There are two ways to turn in assignments: online and on paper. The first week's homework must be turned in *both* online and on paper. (Part of the purpose of this assignment is to make sure you know how to print things; also, **it is from the paper turnin of this homework that the readers make lists of the students they are grading.**) The remaining homework assignments *must* be turned in online, and *may also* be turned in on paper if you would like detailed comments on your work from your reader.

Paper turnin: There are boxes with slots labelled by course in room 283 Soda Hall. (Don't put them in our mailboxes or on our office doors!) What you turn in should include transcripts showing that you have tested your solution as appropriate. **Keep your graded papers** until the semester is over. You may need them in case a grade is entered incorrectly.

Online turnin: You must create a directory (you'll learn how to do that in the lab) with the official assignment name, which will be something like `hw5` or `proj2`. Put in that directory all files you want to turn in. Then, still in that directory, give the shell command `submit hw5` (or whatever the assignment name is). We'll give more details in the lab.

The programming projects must be turned in online as well as on paper in the homework box; the deadline is also noon on the Monday that the project is due. For the group projects, only one member of the group will submit the entire project for the group.

Everything you turn in on paper must show your name(s), your computer account(s), and your section number. Please cooperate about this; make sure they're visible on the *outside* of the paper you turn in, not buried in a comment in a listing. For online submissions, your name(s), computer account(s), and section number must be included in a comment at the beginning of the file.

Webcast of Lectures

For those of you who miss lectures, or wish to review them, they are available online:

`http://webcast.berkeley.edu/courses`

Lost and Found

When people bring us found items from lecture or lab, we take them to the Computer Science office, 387 Soda. If someone from another class finds something you left in Pimentel, it may end up in the College of Chemistry office, 419 Latimer, 642–5882. Another place to check for lost items is the campus police office in Sproul Hall.

Q: I forgot my password. (Or: It won't let me log in.)

Go to the Instructional sys admin staff in 333 Soda, 378 Cory or 386 Cory. Bring your initial class account form or student ID card. Don't ask for another account, especially if you already have some work graded under the old account!

Lecture Outline and Reading Assignments

In the following chart, the readings refer to section numbers in the Abelson and Sussman text. Remember, the reading should be done *before* the week indicated.

week	Monday			Wednesday	Friday	reading
1		**holiday**	functional programming	1/20	1/22	1.1
2	1/25	higher-order procedures	UI (Kay)	1/27	1/29	1.3
3	2/1	UI (Kay)	recursion and iteration	2/3	2/5	1.2.1–4
		Project 1 due Monday, 2/8				
4	2/8	data abstraction, sequences	calculator	2/10	2/12	2.1, 2.2.1
		Midterm Wednesday 2/17, 7–9pm				
5		**holiday**	hierarchical data	2/17	2/19	2.2.2–3, 2.3.1,3
		Project 2 due Monday, 2/22				
6	2/22	interpreter	generic operators	2/24	2/26	2.4–2.5.2
		GCD: 5pm Monday 3/1, MT1, Proj1, HW1–5				
7	3/1	object-oriented programming		3/3	3/5	OOP (reader)
		Midterm Wednesday 3/10, 7–9pm				
8	3/8	assignment, state, environments		3/10	3/12	3.1, 3.2
		Project 3a due Monday, 3/15				
9	3/15	mutable data	vectors	3/17	3/19	3.3.1–3
		spring break				
		Project 3b due Monday, 3/29				
		GCD: 5pm Monday 3/29, MT2, Proj2, HW6–8				
10	3/29	client/server	concurrency	3/31	4/2	3.4
11	4/5	metacircular eval.	analyzing eval.	4/7	4/9	4.1
		Midterm Wednesday 4/14, 7–9pm				
12	4/12	streams	Therac	4/14	4/16	3.5.1–3, 3.5.5,
		Project 4a due Monday, 4/19				Therac
13	4/19	lazy eval.	nondeterministic eval.	4/21	4/23	4.2, 4,3
		Project 4b due Monday, 4/26				
		GCD: 5pm Monday 4/26, MT3, Proj3, HW9–12				
14	4/26	logic programming	review	4/28	4/30	4.4.1–3
15	5/3	RRR Week, no classes		5/5	5/7	
		GCD: 11:30am Tuesday 5/11, Proj4, HW13–14				
		Final Tuesday, 5/11, 11:30–2:30pm				

CS 61A Spring 2010 Week 1

Topic: Functional programming

Lectures: Monday 1/18, Wednesday 1/20, Friday 1/22

Reading: Abelson & Sussman, Section 1.1 (pages 1–31)

Note: With the obvious exception of this first week, you should do each week's reading *before* the Monday lecture. So also start now on next week's reading, Abelson & Sussman, Section 1.3

Homework due noon Tuesday, 1/26:

People who've taken CS 3: Don't use the CS 3 higher-order procedures such as `every` in these problems; use recursion.

1. Do exercise 1.6, page 25. This is an essay question; you needn't hand in any computer printout, unless you think the grader can't read your handwriting. If you had trouble understanding the square root program in the book, explain instead what will happen if you use `new-if` instead of `if` in the `pigl` Pig Latin procedure.

2. Write a procedure `squares` that takes a sentence of numbers as its argument and returns a sentence of the squares of the numbers:

```
> (squares '(2 3 4 5))
(4 9 16 25)
```

3. Write a procedure `switch` that takes a sentence as its argument and returns a sentence in which every instance of the words `I` or `me` is replaced by `you`, while every instance of `you` is replaced by `me` except at the beginning of the sentence, where it's replaced by `I`. (Don't worry about capitalization of letters.) Example:

```
> (switch '(You told me that I should wake you up))
(i told you that you should wake me up)
```

4. Write a predicate `ordered?` that takes a sentence of numbers as its argument and returns a true value if the numbers are in ascending order, or a false value otherwise.

5. Write a procedure `ends-e` that takes a sentence as its argument and returns a sentence containing only those words of the argument whose last letter is E:

```
> (ends-e '(please put the salami above the blue elephant))
(please the above the blue)
```

Continued on next page.

6. Most versions of Lisp provide `and` and `or` procedures like the ones on page 19. In principle there is no reason why these can't be ordinary procedures, but some versions of Lisp make them special forms. Suppose, for example, we evaluate

```
(or (= x 0) (= y 0) (= z 0))
```

If `or` is an ordinary procedure, all three argument expressions will be evaluated before `or` is invoked. But if the variable `x` has the value `0`, we know that the entire expression has to be true regardless of the values of `y` and `z`. A Lisp interpreter in which `or` is a special form can evaluate the arguments one by one until either a true one is found or it runs out of arguments.

Your mission is to devise a test that will tell you whether Scheme's `and` and `or` are special forms or ordinary functions. This is a somewhat tricky problem, but it'll get you thinking about the evaluation process more deeply than you otherwise might.

Why might it be advantageous for an interpreter to treat `or` as a special form and evaluate its arguments one at a time? Can you think of reasons why it might be advantageous to treat `or` as an ordinary function?

Unix feature of the week: `man`

Emacs feature of the week: `C-g`, `M-x apropos`

There will be a "feature of the week" each week. These first features come first because they are the ones that you use to find out about the other ones: Each provides documentation of a Unix or Emacs feature. This week, type `man man` as a shell command to see the Unix manual page on the `man` program. Then, in Emacs, type `M-x` (that's meta-`X`, or `ESC X` if you prefer) `describe-function` followed by the Return or Enter key, then `apropos` to see how the `apropos` command works. If you want to know about a command by its keystroke form (such as `C-g`) because you don't know its long name (such as `keyboard-quit`), you can say `M-x describe-key` then `C-g`.

You aren't going to be tested on these system features, but it'll make the rest of your life a *lot* easier if you learn about them.

CS 61A Spring 2010 Week 1 Lab
Wednesday 1/20 afternoon, Thursday 1/21, or Friday 1/22 morning
Try to get as much done as possible, but don't panic if you don't finish everything.

1. (15 minutes.) Start the Emacs editor, either by typing `emacs` in your main window or by selecting it from the alt-middle mouse menu. (Your TA will show you how to do this.) From the `Help` menu, select the Emacs tutorial. You need not complete the entire tutorial at the first session, but you should do so eventually.

(Parts 2–4: 15 minutes.)

2. Use Emacs to create a file called `pigl.scm` in your directory containing the Pig Latin program shown below:

```
(define (pigl wd)
  (if (pl-done? wd)
      (word wd 'ay)
      (pigl (word (bf wd) (first wd)))))

(define (pl-done? wd)
  (vowel? (first wd)))

(define (vowel? letter)
  (member? letter '(a e i o u)))
```

Make sure you are editing a file whose name ends in `.scm`, so that Emacs will know to indent your code correctly!

3. Now run Scheme by typing meta-S in your Emacs window. You are going to create a transcript of a session using the file you just created, like this:

```
(transcript-on "lab1")      ; This starts the transcript file.
(load "pigl.scm")           ; This reads in the file you created earlier.
(pigl 'scheme)              ; Try out your program.
                            ; Feel free to try more test cases here!
(trace pigl)                ; This is a debugging aid. Watch what happens
(pigl 'scheme)              ; when you run a traced procedure.
(transcript-off)
(exit)
```

4. Use `lpr` to print your transcript file.

Continued on next page.

Week 1 lab continued:

5. (15 minutes.) In the shell, type the command

```
cp ~cs61a/lib/plural.scm .
```

(Note the period at the end of the line!) This will copy a file from the class library to your own directory. Then, using emacs to edit the file, modify the procedure so that it correctly handles cases like `(plural 'boy)`.

6. (20 minutes.) Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

7. (15 minutes.) Write a procedure `dupls-removed` that, given a sentence as input, returns the result of removing duplicate words from the sentence. It should work this way:

```
> (dupls-removed '(a b c a e d e b))
(c a d e b)
> (dupls-removed '(a b c))
(a b c)
> (dupls-removed '(a a a a b a a))
(b a)
```